REMARKS

Reconsideration and allowance of the present application are respectfully requested. Claims 1-36 remain pending in the application. By this Amendment, claims 1, 2, 12, 13, 23 and 24 are amended; and claims 34-36 are added.

In numbered paragraph 2, on page 2 of the Office Action, independent claims 1, 12 and 23, along with various dependent claims, are rejected as being anticipated by U.S. Patent 6,149,318 (Chase et al.). In numbered paragraph 17, on page 7 of the Office Action, dependent claims 7 and 18 are rejected as being unpatentable over the Chase et al. patent, in view of U.S. Patent 5,995,752 (Chao et al.). In numbered paragraph 19, on page 7 of the Office Action, dependent claims 11 and 22 are rejected as being unpatentable over the Chase et al. patent, in view of U.S. Patent 5,987,249 (Grossman et al.). These rejections are respectfully traversed.

Applicants have disclosed exemplary embodiments for dynamically verifying program operation. For example, a method is disclosed for dynamically verifying program operation, comprising: executing a specified computer program; maintaining a shadow array while executing the specified computer program, the shadow array having entries corresponding to respective memory locations used by the specified computer program (see, e.g., specification at page 4, lines 1-4). Each entry of the shadow array indicates a data type of the corresponding respective memory location. While executing the specified computer program, each of a plurality of instructions of the computer program is executed. For the execution of each instruction in a subset of the set of instructions of the computer program, it is determined whether a memory access for executing the instruction is inconsistent with an entry of the shadow array and, if so, a report is generated. The instruction is executed and the

shadow array is updated in accordance with execution of the instruction (see, e.g., specification at page 4, lines 4-10).

The foregoing features are broadly encompassed by claim 1 which recites, among other features, "while executing the specified computer program, maintaining a shadow array, the shadow array having entries corresponding to respective memory locations used by the specified computer program, each entry of the shadow array indicating a data type of the corresponding respective memory location;...determining whether a memory access for executing the instruction is inconsistent with a data type entry of the shadow array and generating a report when an inconsistency is determined."

In one exemplary embodiment, a shadow array is created. Accesses to locations in memory are verified for consistency with the data types stored in the shadow array corresponding to the memory locations (see, e.g., specification at page 3, lines 26-29). The data structure can be a simple table mapping from application memory address to the type/allocation information. Its components can directly correspond to memory addresses. Accordingly, the dynamic verifying of program operation can use a single memory access to the shadow array for type checking based on execution of a binary code without constructing the source code.

The foregoing features are broadly encompassed by independent claims 1, 12 and 23, which also recite "while executing the specified computer program, maintaining a shadow array,...the execution of the specified computer program including executing each of a plurality of instructions of the computer program"; as well as by claims 2, 13 and 24, which also recite, "determining whether execution of

the instruction is inconsistent with the inspected shadow array entry corresponding to the identified memory location."

The dynamic verifying of program operation as disclosed can perform data-type checking on the instruction of a program without requiring a modification of the original instructions. Performing data-type checking encompasses potentially inferring type information from the kind of machine instruction used (e.g., floating point, pointer lookup or integer operation) and potentially inferring type information from the compiler output. As disclosed by the Applicants, the data structure can be constructed after code generation, and can represent types deduced from the compiled binary, and from runtime use of the data being manipulated. Accordingly, a data-type can be checked either by instrumenting an original program, or by interpreting instructions as they are executed (specification at, e.g., page 6, lines 6-10). The foregoing features are broadly encompassed by new dependent claims 34-36.

The documents relied upon by the Examiner, regardless of whether they are considered alone or in the combination discussed by the Examiner, fail to teach or suggest features of claim 1. For example, the Chase et al. patent does not teach or suggest a method of dynamically verifying program operation in which a shadow array is maintained while executing a specified computer program, the method including determining whether execution of the instruction is inconsistent with an entry of the shadow array.

In numbered paragraph 3, page 3 of the Office Action, the Examiner asserts "The table or shadow array is maintained or updated." This assertion is respectfully traversed. The Chase et al. patent does not teach or suggest a shadow array being

maintained while executing a computer program. Rather, the Chase et al. patent discloses comparing "declarations of programming language entities from separately compiled modules" (col. 4, lines 42-47); and deriving a data type from the context of a code function (col. 4, lines 34-41).

For example, the Chase et al. patent discloses that a typeprint is computed from a C or C++ data type (col. 6, lines 14-15). The typeprints are used to compare the data types corresponding to program entities of the software program to be tested in order to determine whether there are multiple data types improperly corresponding to a common give program entity (col. 6, lines 15-19). The type information is deduced as allocations are made. If the allocation is not annotated, no type information is available for it. At least for these reasons, the Chase et al. patent does not teach or suggest a shadow array being maintained while executing a computer program; and the Chase et al. patent does not teach or suggest determining whether a memory access while executing the instruction is inconsistent with a data type entry of the shadow array. Accordingly, the Chase et al. patent does not teach or suggest the features recited in claim 1.

The Chase et al. patent also fails to teach or suggest a shadow array having entries corresponding to respective memory locations used by the specified computer program as recited in claim 1. The Chase et al. patent discloses a data structure with pointers linking different types, e.g., memheader, map, type, etc. (Fig. 9 shows a type tree). As disclosed by the Chase et al. patent, data components do not directly correspond to memory addresses, but rather to "containers" and "types."

The Chase et al. patent also fails to teach or suggest dynamically verifying program operation without modifying an instruction as recited, for example, in new

claims 34-36. The Chase et al. patent discloses a programming language processor performing link-time and run-time error checking of a program written in C or C++ (abstract). The Chase et al. patent discloses modifying a syntax tree with extra annotations. The annotations are transformed and expressed using node types to add run-time error checking and to add code coverage instrumentation (col. 5, lines 4-21).

Further, the Chase et al. patent discloses "an instrumentor inserts calls to the run-time system to enable it to maintain a record of what types and bounds are associated with which addresses, and to query this information for purposes of reporting errors...The instrumentor could be a human programmer, or a source-to-source translator, or a source-to-object translator" (col. 17, lines 20-25). The Chase et al. patent discloses an instrumentor involving manipulation or reading of program source code. As disclosed by the Chase et al. patent, a tree is constructed before the code is generated for the program, whereas the dynamic verifying of program operation as claimed is based on executing a computer program as specified. The Chase et al. patent does not teach or suggest dynamically verifying program operation without modifying an instruction (e.g., as recited in claims 34-36).

The Chao et al. patent, cited in the rejection of claims 7 and 18, does not cure the deficiencies of the Chase et al. patent. The Chao et al. patent was cited for its disclosure of a CPU register (col. 3, lines 17-22). However, the Chao et al. patent does not teach or suggest maintaining a shadow array while executing a specified computer program, each entry of the shadow array indicating a data type of the corresponding respective memory location. The Chao et al. patent also fails to teach or suggest determining whether a memory access while executing the instruction is

inconsistent with a data type entry of the shadow array. The Chao et al. patent and the Chase et al. patent therefore fail to teach or suggest Applicants' combinations of features as recited in independent claim 1. Similar features are recited in independent claims 12 and 23, such that all of Applicants' claims are allowable.

The Grossman et al. patent, cited in the rejection of claims 11 and 22, does not cure the deficiencies of the Chase et al. patent. The Grossman et al. patent was cited for its disclosure of a run time debugging capability, generally for indicating to a user of a run time error condition (col. 18, lines 47-54). The Grossman et al. patent does not teach or suggest maintaining a shadow array while executing the specified computer program, each entry of the shadow array indicating a data type of the corresponding respective memory location. The Grossman et al. patent also fails to teach or suggest determining whether a memory access for executing the instruction is inconsistent with a data type entry of the shadow array. The Grossman et al. patent and the Chase et al. patent therefore fail to teach or suggest Applicants' claim 1 combination of features. Claims 12 and 23 recite similar features, such that all of Applicants' claims are allowable.

Even if the cited documents could have been combined in the manner asserted by the Examiner, the combination would not have resulted in the presently claimed invention. The Chase et al. patent, the Chao et al. patent and the Grossman et al. patent, individually or in combination, do not teach or suggest a method of dynamically verifying program operation. These documents, regardless of whether they are considered individually or in combination, do not teach or suggest maintaining a shadow array while executing a specified computer program; and "determining whether a memory access for executing the instruction is inconsistent

with a data type entry of the shadow array and generating a report when an inconsistency is determined," as recited in claim 1. Independent claims 12 and 23 are directed to a computer program product reciting similar features.

For the foregoing reasons, Applicants' independent claims 1, 12 and 23 are therefore also allowable. The remaining claims recite additional advantageous features which further distinguish over the documents relied upon by the Examiner. For example, the applied references, individually or in combination, do not teach or suggest "determining whether proper execution of the instruction requires accessing data of a predefined data type that is different from the data type specified by the entry of the shadow array," as recited in claim 5; do not teach or suggest "determining whether proper execution of the instruction is inconsistent with the data type specified by the entry of the shadow array," as recited in claim 6; and do not teach or suggest "compiling a source code program into the specified computer program; obtaining debugging information related to the specified computer program; and initializing the shadow memory based on the debugging information," as recited in claim 10.

All objections and rejections raised in the Office Action having been addressed, it is respectfully submitted that the application is in condition for allowance and a Notice of Allowance is respectfully solicited.

Respectfully submitted,

BURNS, DOANE, SWECKER & MATHIS, L.L.P.

Date: September 14, 2005

By: for Reg. No. 48,360
Patrick C. Keane
Registration No. 32,858

P.O. Box 1404
Alexandria, Virginia 22313-1404
(703) 836-6620